**Brock University Chat Bot**

COSC 4P02 Project

**PROGRESS REPORT 2**

Members:

Tirth Mehta (Leader) (6039812)

Harveer Khangura (6052674)

Matthew Snow (6051981)

Jose Valdez (5941323)

Mazin Bamohrez (6157861)

Marlon Ramnarais (6039275)

**1.0 Backend - Chatbot NLP Action and Response**

**1.1 Description**

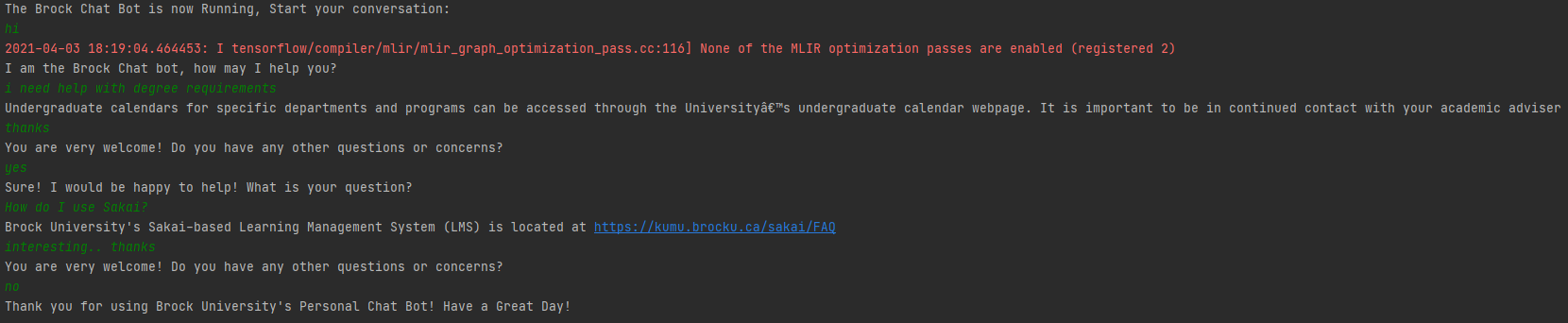
The development of the NLP algorithm and training was complete during the second part of our first iteration. The NLP allows for creating simple, yet, explicit chat conversations with the bot and allows for specific inputs as data and to return a specific response from a set of responses. The data is pushed through a JSON file called ‘faq-intents.json’ at which the file data is separated by intents (tags(patterns, responses)). A string pattern is given to the chatbot through an array of similar/related words such as (Hi, Hello, Hey, Good day) at which the tag is determined. From certain tags, the response can be calculated by the NLP and these recurring calculations will be used as AI training. Therefore, a user can input a greeting, or ask about something such as degree requirements for their program at Brock University, and the Chatbot can query/calculate an accurate response.

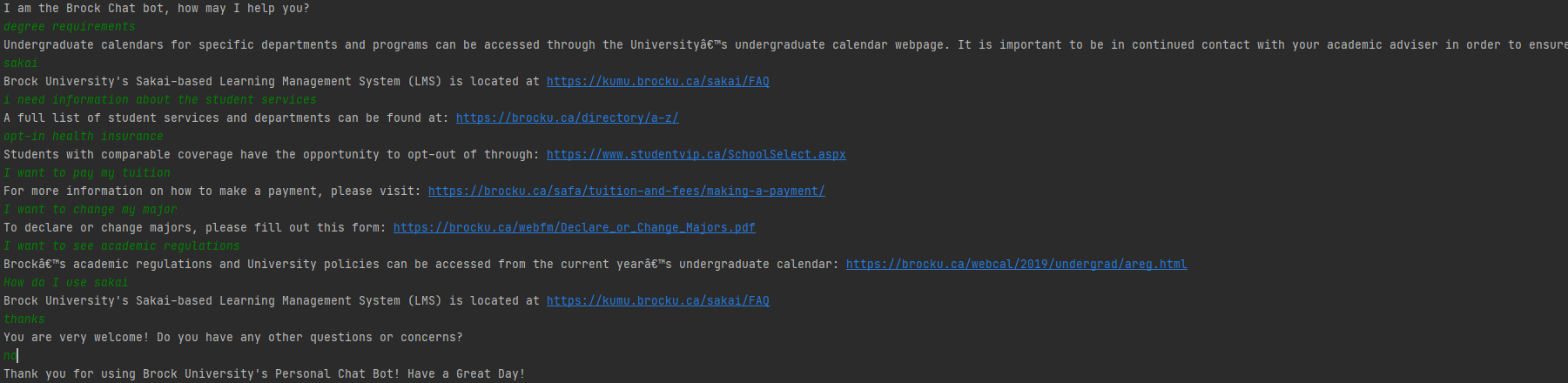
Although, this data is not static and is adaptable to how the chatbot can read and calculate different pattern types and would have the fluidity and flexibility to input combinations of patterns such as “Hello, How are you?”.

There are two files at which the Chatbot will first be trained, and the other for simply using the chatbot to determine the results of the training. The **Chatbot-Training.py** file includes the chatbot training algorithm to feed in data from the JSON file and inquire the ability to recognize what information the user is providing. By using the tensorflow and Natural Language Toolkit (NLTK) toolkits, the training class results in the measurement of ‘Accuracy’ for each user input and their respective ‘tag’ within the JSON dataset.

The **Chatbot-Main.py** file includes three main functional methods that categorize and maintain the user input from the console and pulls a response that would be random in respect to its related ‘tag’ and ‘pattern.’

**Sample Output from Console** :





**1.2 Takeaways and Future Plans**

Error handling and User error - This functionality in the backend is still under development as the main purpose of this portion of the iteration is to successfully train and categorize user input and chatbot output data. The plan to have the error handling placed in the backend will be to ensure spelling errors, invalid inputs are handled correctly. We plan to have the integration with the front end for the upcoming iteration and ensure this functionality is set.

The backend still needs to be implemented for both the Frontend and Backend for storing Chat interactions and metadata of the User. We plan to have the database integrated ultimately within the frontend and backend to ensure logging of data is correctly placed. Each chat interaction will include an id, with each chat entry enumerated with a timestamp.

**1.3 Backend Unit Testing**

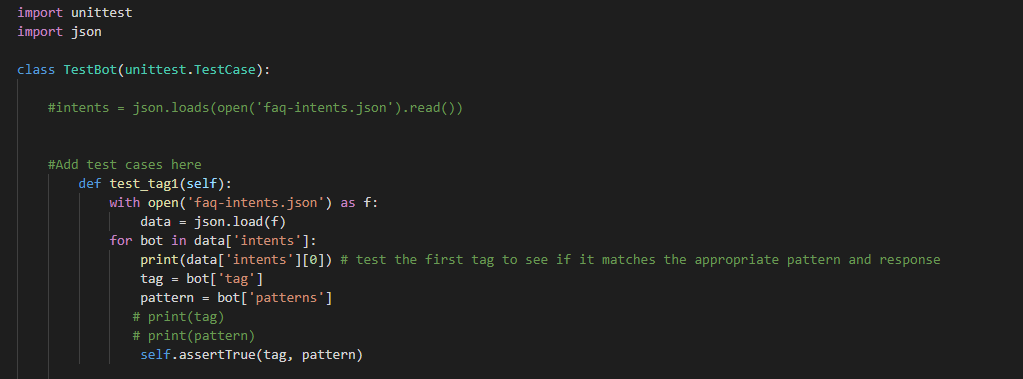
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Name** | **Input** | **Output** | **Expected Output** | **Results** |
| **JSON feed test**  **(backend)** | **Tag = ‘Greetings’**  **Pattern = “Hello”, “Hi”, “Hey”, “ Greetings”, “hi”, “hey”** | **Ok** | **Ok** | **Working as expected** |
| **Chatbot Interaction test (1) - User Input**  **Question (SAKAI)** | **“Hi”**  **“How do I use Sakai”** | **Sakai Response(s) from JSON File** | **Sakai Response(s) from JSON File** | **Working as expected** |
| **Chatbot Interaction test (2) - User Input**  **Question (Degree Requirements)** | **“Hi”**  **“How do I access my Degree Requirements”** | **Sakai Response(s) pulled from JSON File** | **Sakai Response(s) pulled from JSON File** | **Working as expected** |
| **Interaction test (3) - User Multiple User Inputs (SAKAI, Degree Requirements)** | **“Hello”**  **“Where can I use sakai”**  **“How do I access degree requirements”** | **Sakai Response(s) pulled from JSON File** | **Sakai Response(s) pulled from JSON File** | **Working as expected** |
|  |  |  |  |  |
|  |  |  |  |  |

**1.4 Unit testing backend results**

We implemented the **test\_bot.py** as a unit testing class where we have individual functions responsible for testing specific parts of the backend implementation. In this testing class, we specifically tested that whatever is being fed into the training class from the json files matches the tags and patterns. This ensures the appropriate responses are made when the user asks a question based on the word patterns the bot will analyze. For example, in the json file we have a tag that relates to a specific word pattern. We used the assertTrue function from the unittest library to ensure each pattern matches the tag. The results were successful with no errors as shown below



Unit test sample



**2.0 Front end User connectivity and Collection**

**2.1 Description** -

To begin, the front end has seen a large overhaul compared to it’s first iteration when it was using socket.io and as of now all usage of socket.io has been removed as it initially caused issues with attempting to get multiple unique sessions running. A replacement system using a mix of JavaScript (AJAX) and built in flask method called request allows us to get user input with no issues. Messages are sent between the html document and python in the format of JSON data. The method “def handle\_intput():” handles the user input when the submit button is pressed, the HTML updates the url to end with “/join” and this causes the app.py to react and take the user input and provide response all of which is converted JSON, processed and return JSON. The HTML displays the user message and the ChatBot response.

Sessions/cookies are stored on the browser side of all the interactions between the user and the chatbot. This way we can implement a command to the system to display the users information with the system as well as the chat bot’s responses from their previous interactions.

Unique sessions have been implemented as well in which accessing the site from different browsers will allow for unique conversation with chatbot.

**2.2 Front end Backlog and Future Release**

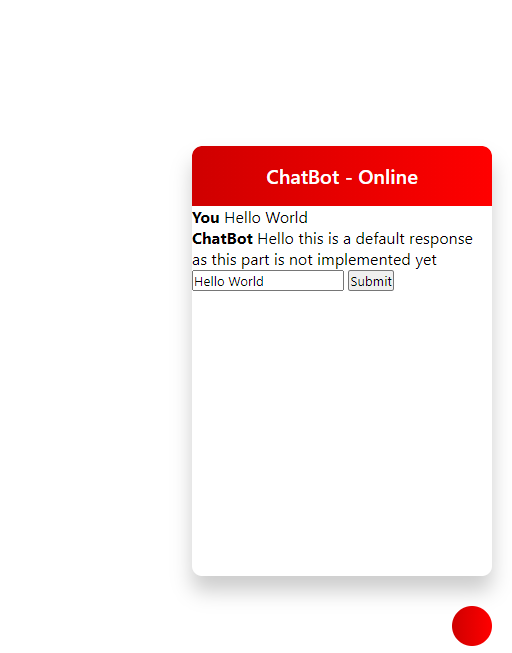
To continue, the backlog which was not completed before progress report 2 is due contains the following objectives. Commands which include displaying user information, displaying the user chat logs, displaying a help command in which the user is able to see all other commands. The backlog also contains full unit testing of unique sessions as that was implemented very recently as such is a part of the backlog to be completed. Integration between the front end and the UI has been started but cannot be considered as completed but as one of the current working objectives as such it will be considered a backlog item to be completed.

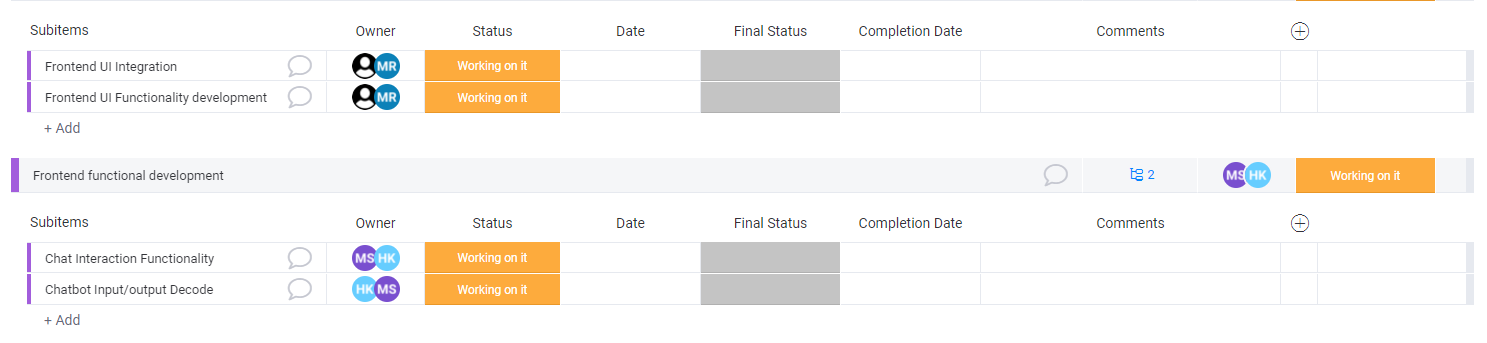
Future releases will include additional commands aside from the 3 contained within the backlog. The full integration with the backend so we are able to get smooth conversation going with a user with a user inputting a string and receiving a message that is based on the user string. Proper unit testing methods for all new implementations.

**2.3 Testing**

To start, all the items currently implemented such as handle\_input method and sessions method can not be unit tested as they run through built in flask and have been tested through other means such as constant usage and random inputs. With the new methods such as interpretMessage(msg) and unique sessions unit testing will become necessity for the next few iterations as project is coming to a close soon we will be focusing more completing the backlog and ensuring stability through the use of unit testing

-Below is a current snippet of the HTML and front end being combined. This is not indicative of what the final product will look like.

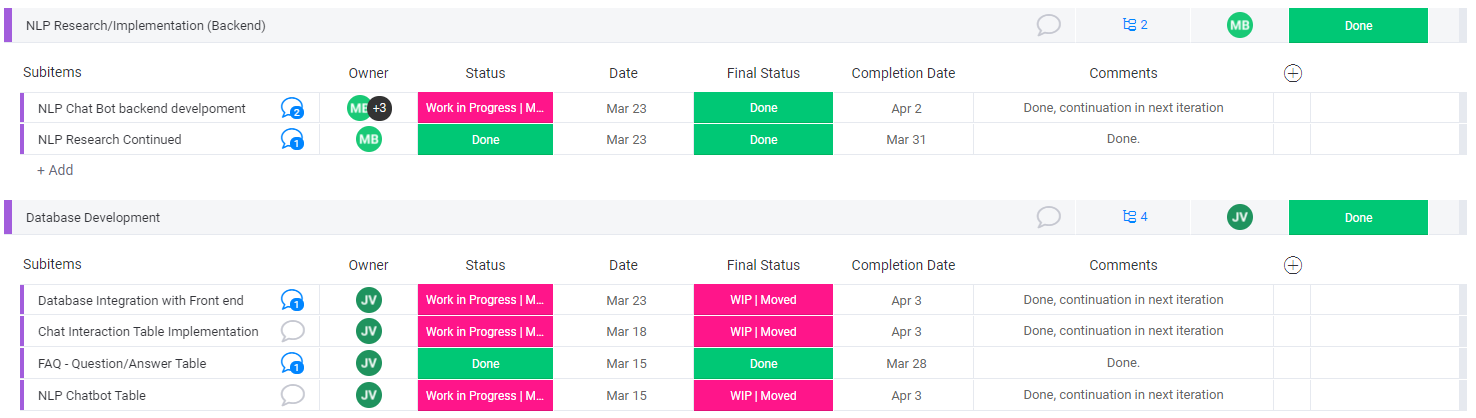


****

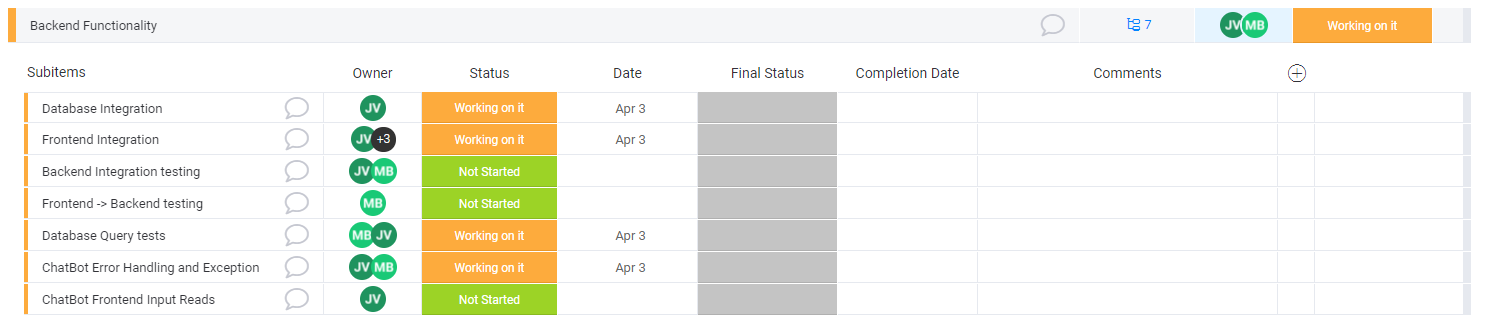
**Iteration: Sprints and Backlog**

**Backend** **Status**

All of the primitive and important User Stories which are needed for future implementation within the next Iteration have been completed. The database development has been completed however, integration between the frontend and backend is the next step. We look to conduct integration testing and database query (testing) in the next/final iteration and is prioritized and the next sprint.



**Backend User Stories** (Backlog)



**3.0 HTML//CSS - UI Layout and Design**

**3.1 Description**

During sprint 1, we reworked our design by implementing a multi-page form. We redesigned the welcoming menu for what a user must input to begin. A user will now input their name, email as well as their inquiry before beginning a chat session. Once they input all required fields and click “start chat”, it will redirect them over to a chat window within the widget where the user and chatbot will converse. Here we started implementing a javascript file to handle the redirecting feature to allow for a multi-page widget.

**3.2 Takeaways and Future Plans**

In the next sprint, we will implement more UI functions and designs that make the chatbot more user-friendly for the users. The chat itself will offer a scroll-bar option where the user can look back at their conversations and keep track of the answered questions. We will also implement the “Enter” key as another option rather than the user having to click “Submit”. Users are used to using the “Enter” key as another form of confirm making this more user-friendly. We will also remove the Submit button from the bottom of the text-box and add it into the text-box to provide more room for conversation and provide a cleaner UI.

**3.3 Testing:**

We will continue testing different types of mocks and designs to make the User Interface as friendly as possible. Besides giving useful information, the chatbot must be created in a way where the user can easily find the buttons and information they need. We will work on finding more shortcuts (Enter key for submit) to test the different ways the chatbot can be optimized for the up-most User Experience.